**BANK MANAGEMENT SYSTEM**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology/Master of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**Mythili Nagavalli Nandamuri AP23110010543**

**Mukhtaar Ahmed Shaik AP23110010554**

**Veda Rushitha Bonam AP23110010539**

**Joyson Thadikonda AP23110010529**

**A picture containing text

Description automatically generated**

Under the Guidance of

**Dr. Kavitha Rani Karnena**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**November, 2024**

# Certificate

Date: 20-Nov-24

This is to certify that the work present in this Project entitled “**BANK MANAGEMENT SYSTEM**” has been carried out by **Mythili Nagavalli Nandamuri, Veda Rushitha Bonam, Mukhtaar Ahmed Shaik and Joyson Thadikonda** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

**Supervisor**

**Dr. Kavitha Rani Karnena**

**Signature**

**Prof. / Dr. [Kavitha Rani Karnena]**

# Acknowledgements

We take immense pleasure in presenting our C++ project, the **Bank Management System**, which showcases the collective effort, dedication, and guidance that contributed to its successful completion.

First and foremost, we would like to express our heartfelt gratitude to our professor and project guide, **Prof. Kavitha Rani Karnena**, for her invaluable support, encouragement, and expert insights throughout this journey. Her mentorship and constructive suggestions were vital in helping us overcome challenges and shape this project into its final form.

We also deeply thank our entire team for their hard work, commitment, and collaboration. Each member's unique contributions and unwavering dedication were instrumental in turning our vision into reality. The mutual understanding and support within the team ensured smooth progress at every stage of the project.

This project has provided us with a tremendous learning experience, allowing us to apply C++ concepts practically while enhancing our problem-solving and teamwork skills. We are grateful for the opportunity to work under such excellent guidance and alongside such a dedicated team.

Thank you to everyone who contributed their time and effort to make this project a success and a memorable experience.

# Table of Contents

[Certificate i](#_gjdgxs)

[Acknowledgements ii](#_30j0zll)

[Table of Contents](#_1fob9te) iii

[Abstract iv](#_3znysh7)

[1.](#_17dp8vu) Introduction 1

[2.](#_lnxbz9) Methodology 3

[3.](#_44sinio) Implementation 5

[4.](#_2jxsxqh) Outputs 7

[5.](#_z337ya) Concluding Remarks 9

# Abstract

The Bank Management System is a C++ application designed to facilitate basic banking operations securely and efficiently. The system utilizes object-oriented programming principles to organize functionality into distinct classes: ‘Transaction’, ‘Account’, and ‘Bank’. This modular design enables seamless account management, transaction tracking, and secure operations.

The application supports key features such as deposit, withdrawal, and fund transfer, while ensuring transaction integrity through password-protected accounts. Users can view detailed transaction histories for auditing purposes. By leveraging dynamic data structures like vectors, the system efficiently manages multiple accounts and transactions.

The project serves as a foundational model for banking applications, with potential enhancements such as persistent data storage, advanced security measures, and a graphical user interface to improve usability and scalability. This system exemplifies the principles of robust software design and is ideal for educational purposes or as a prototype for more complex banking solutions.

# 1.Introduction

The Bank Management System is a software application developed in C++ to simulate the core functionalities of a banking system. Designed with an object-oriented approach, the system effectively encapsulates the complexities of managing accounts, transactions, and banking operations into modular components. It provides users with essential banking features, including account creation, deposits, withdrawals, fund transfers, and transaction history management.

In today's digital world, efficient and secure management of financial transactions is crucial for any banking institution. This system demonstrates the implementation of these operations in a simplified yet scalable manner. By integrating principles of data encapsulation, inheritance, and abstraction, the program ensures maintainability and extensibility.

The system also emphasizes security by incorporating password-protected accounts to prevent unauthorized access. While the current implementation focuses on fundamental banking operations, it lays a strong foundation for incorporating advanced features like data persistence, interest calculations, and user authentication.

This project serves as an excellent learning tool for understanding object-oriented programming concepts and their practical application in financial systems. It highlights how programming can address real-world challenges through structured and efficient design.

# 2. Methodology

The development of the Bank Management System follows a structured and systematic approach to implement core banking functionalities. The methodology comprises several stages, from planning to implementation, with a focus on modularity, security, and user interaction. Below are the key steps involved in the methodology:

**1. System Design**

The system is designed using an **object-oriented programming (OOP)** paradigm, which allows for modularity and scalability. Key entities in the banking process, such as accounts and transactions, are modeled as classes with attributes and methods that represent their real-world behavior.

* **Class Design**:
  + Transaction: Represents individual transactions with attributes for type (deposit/withdrawal/transfer), amount, and timestamp.
  + Account: Handles account-specific operations, such as balance management, password verification, and transaction history.
  + Bank: Serves as a manager for all accounts and oversees account creation, fund transfers, and searches.

**2. Development Workflow**

The system is implemented incrementally, ensuring that each component is functional before integration.

* **Step 1 : Class Implementation**  
  Each class is developed with its attributes and methods, starting with Transaction, then Account, and finally Bank.
* **Step 2 : User Interface Design**  
  A command-line interface is built to allow user interaction with the system. This interface enables operations like creating accounts, depositing or withdrawing funds, transferring money, and viewing transaction histories.
* **Step 3 : Error Handling and Validation**  
  Password verification, sufficient balance checks, and proper input handling are incorporated to ensure the system is robust and secure.

**3. Data Flow**

The system uses a hierarchical data flow:

1. **User Interaction**: Input is taken from the user via the console.
2. **Account Lookup**: The Bank class locates the relevant account using its account number.
3. **Transaction Execution**: Once an account is verified, operations like deposit, withdrawal, or transfer are performed.
4. **Transaction Logging**: Successful operations are recorded in the account’s transaction history.

**4. Security Measures**

Password-protected accounts ensure that sensitive information and operations are accessible only to authorized users. Passwords are validated using methods in the Account class, and sensitive operations like viewing transaction history require password verification.

**5. Testing and Validation**

The system is rigorously tested for:

* Edge cases, such as insufficient balance or invalid input.
* Security flaws in password management.
* Scalability with multiple accounts and transactions.

**6. Extensibility**

The program is designed to be extensible, allowing future enhancements such as:

* Persistent data storage for saving account details and transactions.
* Integration of additional features like interest calculation and loan management.
* A graphical user interface for better user experience.

# 3. Implementation

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Transaction {

public:

string type;

double amount;

string timestamp;

Transaction(const string& t, double a, const string& ts)

: type(t), amount(a), timestamp(ts) {}

};

class Account {

private:

string accountNumber;

string password;

double balance;

vector<Transaction> transactions;

public:

Account(const string& accNum, const string& pwd)

: accountNumber(accNum), password(pwd), balance(0.0) {}

bool verifyPassword(const string& pwd) const {

return password == pwd;

}

void deposit(double amount) {

balance += amount;

transactions.emplace\_back("Deposit", amount, "Timestamp");

}

bool withdraw(double amount) {

if (amount > balance) {

cout << "Insufficient funds." << endl;

return false;

}

balance -= amount;

transactions.emplace\_back("Withdrawal", amount, "Timestamp");

return true;

}

void transfer(Account& other, double amount) {

if (withdraw(amount)) {

other.deposit(amount);

transactions.emplace\_back("Transfer", amount, "Timestamp");

}

}

void printTransactions(const string& pwd) const {

if (!verifyPassword(pwd)) {

cout << "Invalid password. Access denied." << endl;

return;

}

cout << "Transactions for account " << accountNumber << ":\n";

for (const auto& txn : transactions) {

cout << txn.timestamp << ": " << txn.type << " of $" << txn.amount << endl;

}

cout << "Remaining balance: $" << balance << endl;

}

double getBalance() const {

return balance;

}

const string& getAccountNumber() const {

return accountNumber;

}

};

class Bank {

private:

vector<Account> accounts;

public:

void createAccount(const string& accNum, const string& pwd) {

accounts.emplace\_back(accNum, pwd);

cout << "Account " << accNum << " created successfully." << endl;

}

Account\* findAccount(const string& accNum) {

for (auto& acc : accounts) {

if (acc.getAccountNumber() == accNum) {

return &acc;

}

}

cout << "Account not found." << endl;

return nullptr;

}

};

int main() {

Bank bank;

int choice;

string accNum;

string toAccNum;

string password;

while (true) {

cout << "1. Create Account\n2. Deposit\n3. Withdraw\n4. Transfer\n5. View Transactions\n6. Exit\n";

cout << "Choose an option: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter account number: ";

cin >> accNum;

cout << "Set your password: ";

cin >> password;

bank.createAccount(accNum, password);

break;

case 2:

cout << "Enter account number: ";

cin >> accNum;

{

Account\* acc = bank.findAccount(accNum);

if (acc) {

double amount;

cout << "Enter amount to deposit: ";

cin >> amount;

acc->deposit(amount);

}

}

break;

case 3:

cout << "Enter account number: ";

cin >> accNum;

{

Account\* acc = bank.findAccount(accNum);

if (acc) {

double amount;

cout << "Enter amount to withdraw: ";

cin >> amount;

acc->withdraw(amount);

}

}

break;

case 4:

cout << "Enter your account number: ";

cin >> accNum;

{

Account\* acc = bank.findAccount(accNum);

if (acc) {

cout << "Enter recipient account number: ";

cin >> toAccNum;

Account\* toAcc = bank.findAccount(toAccNum);

if (toAcc) {

double amount;

cout << "Enter amount to transfer: ";

cin >> amount;

acc->transfer(\*toAcc, amount);

}

}

}

break;

case 5:

cout << "Enter account number: ";

cin >> accNum;

{

Account\* acc = bank.findAccount(accNum);

if (acc) {

cout << "Enter your password for verification: ";

cin >> password;

acc->printTransactions(password);

}

}

break;

case 6:

cout << "Exiting..." << endl;

return 0;

default:

cout << "Invalid option. Please try again." << endl;

}

}

}

# 4. Outputs:

The output of the code :

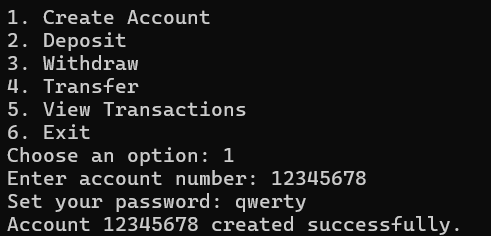
* When the code runs, the following is displayed

A screenshot of a computer screen

Description automatically generated

The following are the cases when the code displayed :

**CASE 1 :** In order to create an account the user selects option 1



**CASE 2 :** In order to deposit, the user selects the 2nd option.

**A screen shot of a computer

Description automatically generated**

**CASE 3 :** To withdraw the amount from the account, choose option 4.

**A screen shot of a computer

Description automatically generated**

**CASE 4 :** To transfer the money from one account to another, the user has to create a new account to which the money needs to be transferred and then transfer the amount.

A screenshot of a computer

Description automatically generated

**CASE 5 :** To check all the bank transfers, the user has to choose 5.

**A screenshot of a computer screen

Description automatically generated**

**CASE 6 :** User selects 6 to exit.

**A black screen with a black background

Description automatically generated**

# Concluding Remarks

The Bank Management System implemented in C++ successfully demonstrates a robust and scalable approach to managing basic banking operations such as account creation, deposits, withdrawals, fund transfers, and transaction history tracking. By leveraging object-oriented principles, the system encapsulates functionalities within modular classes, ensuring reusability, security, and ease of maintenance.

The project emphasizes secure account management through password protection and dynamic transaction logging, providing users with a reliable banking experience. Its well-structured design lays the groundwork for future enhancements, including data persistence, advanced user authentication, and additional financial services.

This system serves as a practical application of programming concepts and a foundation for more complex banking systems. It is well-suited for educational purposes and as a prototype for real-world implementations, demonstrating how software solutions can efficiently address real-world challenges.